卡中心集采培训

Touda培训

- 微服务治理中心
- 服务注册中心

Touda规范的理解

- ToudaIDE配置规范及要求
- 新增微应用规范及流程
- 渠道服务业务码长度不能超过5位；渠道服务描述；
- 查询交易及无事务要求的交易流，在交易流配置中要选择无事务；
- 应用申请、服务申请发布流程；
- 调用方、提供方、注册中心、服务治理
- 能够注册的服务必须是在注册中心已经记录在册的服务；前提是：服务提供方申请过；调用方同样也必须申请过并且被允许调用。

Touda3.0新特性

服务阻断、降级

- 熔断器功能：降级逻辑：**断路器打开**、**线程池满**是降级的两个重要指标
- 灰度发布：调用方在程序启动的时候首先从配置中心拉取配置：app.evn；app.version;灰度标识（调用灰度）；配置中心灰度配置项；在配置中心配置灰度规则；服务级别、应用级别、服务器级别、环境级别；
- 新增Kafka渠道：创建自定义渠道、实现扩展类、添加相应的配置信息；
- 配置中心：为微应用集中管理配置项
- Touda-DFS;临时文件只存储30天；海量小文件存储、高并发高性能支持；文件唯一标识。

重点知识点

- 微服务概念；
- Touda平台；SEDA、无前端展示；
- 去中心化、端对端网络；
- Touda服务相关开发规范；
- 服务发布申请流程（**重点**）；
- 新增微应用流程；
- 发布申请；容易出错的地方；模板填写；
- 应用码不超过5位及交易描述；
- 交易服务的三个功能；

- 服务发布流程：投产之前服务流程已经走到**等待投产（上生产之前流程必须走完）**
- 应用ID的格式
- 模板的易错点：（截图）
- 日志文件及每个日志记录的文件内容

Touda3.0新特性

- 熔断器降级逻辑的两个指标；
- 灰度发布场景（了解）；验证特性；
- 配置中心特性
- Touda-DFS：文件定位、主要业务场景。

开发规范

- 01-Touda应用系统新增申请表V1.0.doc
- 02-Touda服务发布申请表V1.0.xlsx
- 03-非Touda服务调用申请表V1.0.xlsx
- 04-Touda服务变更申请表V1.0.xlsx
- ToudaCache-20180614.pdf
- ToudaDFS-20180614.pdf
- Touda日志说明*V1.0.0.doc*
- Touda配置文件说明.docx
- Touda开发配置规范V0.0.1.docx
- Touda项目创建规范V1.0.1.docx
- Touda代理使用说明*V1.0.0.docx*
- Touda平台开发规范*V1.0.1.doc*

- Touda项目创建规范*V1.0.1.docx*
- Touda服务代理使用手册-V0.0.1.docx
- Touda配置中心说明文档*V1.1.docx*

int 和 Integer

int 是基本数据类型,**Integer**是其**包装类**，注意是一个类。为什么要提供包装类呢？一是为了在各种类型间转化，通过各种方法的调用。否则你无法直接通过变量转化。比如，现在int要转为String：

```
int a=0;
String result=Integer.toString(a);
```

在java中包装类，比较多的用途是用在于各种数据类型的转化中。

Demo：

```
//通过包装类来实现转化的
int num=Integer.valueOf("12");
int num2=Integer.parseInt("12");
double num3=Double.valueOf("12.2");
double num4=Double.parseDouble("12.2");
//其他的类似。
//通过基本数据类型的包装来的valueOf和parseXX来实现String转为XX
//这里括号中几乎可以是任何类型
String a=String.valueOf("1234");
String b=String.valueOf(true);
//通过包装类的toString()也可以
String c=new Integer(12).toString();
String d=new Double(2.3).toString();
```

再举例下。比如我现在要用泛型

```
List<Integer> nums;
```

这里<>需要类。如果你用int。它会报错的。

重载和重写

Override（重写）

1. 方法名、参数、返回值相同。
2. 子类方法不能缩小父类方法的访问权限。
3. 子类方法不能抛出比父类方法更多的异常(但子类方法可以不抛出异常)。
4. 存在于父类和子类之间。
5. 方法被定义为final不能被重写。

Overload（重载）

1. 参数类型、个数、顺序至少有一个不相同。
2. 不能重载只有返回值不同的方法名。
3. 存在于父类和子类、同类中。

final, finally, finalize 的区别

- final修饰符（关键字）如果一个类被声明为final，意味着它不能再派生出新的子类，不能作为父类被继承。因此一个类不能既被声明为 abstract的，又被声明为final的。将变量或方法声明为final，可以保证它们在使用中不被改

变。被声明为final的变量必须在声明时给定初值，而在以后的引用中只能读取，不可修改。被声明为final的方法也同样只能使用，不能重载。

- finally在异常处理时提供 finally 块来执行任何清除操作。如果抛出一个异常，那么相匹配的 catch 子句就会执行，然后控制就会进入 finally 块（如果有的话）。
- finalize方法名。Java 技术允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在确定这个对象没有被引用时对这个对象调用的。它是在 Object 类中定义的，因此所有的类都继承了它。子类覆盖 finalize() 方法以整理系统资源或者执行其他清理工作。finalize() 方法是在垃圾收集器删除对象之前对这个对象调用的。

封装、集成、多态和抽象

封装

封装给对象提供了隐藏内部特性和行为的能力。对象提供一些能被其他对象访问的方法来改变它内部的数据。在 Java 当中，有 3 种修饰符： public， private 和 protected。每一种修饰符给其他的位于同一个包或者不同包下面对象赋予了不同的访问权限。下面列出了使用封装的一些好处：

- 通过 隐藏对象的属性来保护对象内部的状态。
- 提高了代码的可用性和可维护性，因为对象的行为可以被单独的改变或者是扩展。
- 禁止对象之间的不良交互提高模块化

继承

继承给对象提供了从基类获取字段和方法的能力。继承提供了代码的重用行，也可以在不修改类的情况下给现存的类添加新特性。

多态

多态是编程语言给不同的底层数据类型做相同的接口展示的一种能力。一个多态类型上的操作可以应用到其他类型的值上面。

抽象

抽象是把想法从具体的实例中分离出来的步骤，因此，要根据他们的功能而不是实现细节来创建类。 Java 支持创建只暴漏接口而不包含方法实现的抽象的类。这种抽象技术的主要目的是把类的行为和实现细节分离开。

基础知识点汇编

数组有没有length()方法？String有没有length()方法？

数组没有length()方法，有length 的属性。String 有length()方法。JavaScript中，获得字符串的长度是通过length属性得到的，这一点容易和Java混淆。

构造器（constructor）是否可被重写（override）？

构造器不能被继承，因此不能被重写，但可以被重载。

两个对象值相同(x.equals(y) == true)，但却可有不同的hash code，这句话对不对？

不对，如果两个对象x和y满足x.equals(y) == true，它们的哈希码（hash code）应当相同。

Java对于eqauls方法和hashCode方法是这样规定的：

- 如果两个对象相同（equals方法返回true），那么它们的hashCode值一定要相同；
- 如果两个对象的hashCode相同，它们并不一定相同。当然，你未必要按照要求去做，但是如果你违背了上述原则就会发现在使用容器时，相同的对象可以出现在Set集合中，同时增加新元素的效率会大大下降（对于使用哈希存储的系统，如果哈希码频繁的冲突将会造成存取性能急剧下降）。

是否可以继承String类?

String 类是final类，不可以被继承。

补充：继承String本身就是一个错误的行为，对String类型最好的重用方式是关联关系（Has-A）和依赖关系（Use-A）而不是继承关系（Is-A）。

当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递?

是值传递。Java语言的方法调用只支持参数的值传递。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的属性可以在被调用过程中被改变，但对对象引用的改变是不会影响到调用者的。C++和C#中可以通过传引用或传输出参数来改变传入的参数的值。在C#中可以编写如下所示的代码，但是在Java中却做不到。

抽象类和接口

- 接口是公开的，里面不能有私有的方法或变量，是用于让别人使用的，而抽象类是可以有私有方法或私有变量的，
- 另外，实现接口的一定要实现接口里定义的所有方法，而实现抽象类可以有选择地重写需要用到的方法，一般的应用里，最顶级的是接口，然后是抽象类实现接口，最后才到具体类实现。
- 还有，接口可以实现多重继承，而一个类只能继承一个超类，但可以通过继承多个接口实现多重继承，接口还有标识（里面没有任何方法，如Remote接口）和数据共享（里面的变量全是常量）的作用。

反射的用途及实现

Java反射机制主要提供了以下功能：在运行时构造一个类的对象；判断一个类所具有的成员变量和方法；调用一个对象的方法；生成动态代理。反射最大的应用就是框架。

Java反射的主要功能：

- 确定一个对象的类
- 取出类的modifiers,数据成员,方法,构造器,和超类.
- 找出某个接口里定义的常量和方法说明.
- 创建一个类实例,这个实例在运行时刻才有名字(运行时间才生成的对象).
- 取得和设定对象数据成员的值,如果数据成员名是运行时刻确定的也能做到.
- 在运行时刻调用动态对象的方法.
- 创建数组,数组大小和类型在运行时刻才确定,也能更改数组成员的值.

反射的应用很多，很多框架都有用到：

- spring 的 ioc/di 也是反射….
- javaBean和jsp之间调用也是反射….
- struts的 FormBean 和页面之间…也是通过反射调用….
- JDBC 的 classForName()也是反射…..
- hibernate的 find(Class clazz) 也是反射….

session 与 cookie

cookie 是 Web 服务器发送给浏览器的一块信息。浏览器会在本地文件中给每一个 Web 服务器存储cookie。以后浏览器在给特定的 Web 服务器发请求的时候，同时会发送所有为该服务器存储的cookie。

下面列出了session和cookie的区别：

无论客户端浏览器做怎么样的设置，session都应该能正常工作。客户端可以选择禁用cookie，但是，session仍然是能够工作的，因为客户端无法禁用服务端的session。

equals 与 == 的区别

值类型（int,char,long,boolean等）都是用==判断相等性。对象引用的话，==判断引用所指的对象是否是同一个。equals是Object的成员函数，有些类会覆盖（override）这个方法，用于判断对象的等价性。例如String类，两个引用所指向的String都是"abc"，但可能出现他们实际对应的对象并不是同一个（和jvm实现方式有关），因此用==判断他们可能不相等，但用equals判断一定是相等的。

集合详解

List 和 Set 区别

List,Set都是继承自Collection接口。

List特点： 元素有放入顺序，元素可重复

Set特点：元素无放入顺序，元素不可重复，重复元素会覆盖掉

注意：元素虽然无放入顺序，但是元素在set中的位置是有该元素的HashCode决定的，其位置其实是固定的，加入Set 的Object必须定义equals()方法 ，另外list支持for循环，也就是通过下标来遍历，也可以用迭代器，但是set只能用迭代，因为他无序，无法用下标来取得想要的值。

Set和List对比：

- Set：检索元素效率低下，删除和插入效率高，插入和删除不会引起元素位置改变。
- List：和数组类似，List可以动态增长，查找元素效率高，插入删除元素效率低，因为会引起其他元素位置改变。

List 和 Map 区别

- List是对象集合，允许对象重复。
- Map是键值对的集合，不允许key重复。

ArrayList 与 LinkedList 区别

ArrayList：

- 优点：ArrayList是实现了基于动态数组的数据结构,因为地址连续，一旦数据存储好了，查询操作效率会比较高（在内存里是连着放的）。
- 缺点：因为地址连续， ArrayList要移动数据,所以插入和删除操作效率比较低。

LinkedList：

- 优点：LinkedList基于链表的数据结构,地址是任意的，所以在开辟内存空间的时候不需要等一个连续的地址，对于新增和删除操作add和remove，LinedList比较占优势。LinkedList 适用于要头尾操作或插入指定位置的场景
- 缺点：因为LinkedList要移动指针,所以查询操作性能比较低。

适用场景分析：

当需要对数据进行对此访问的情况下选用ArrayList，当需要对数据进行多次增加删除修改时采用LinkedList。

ArrayList 与 Vector 区别

```
//构造一个具有指定初始容量的空列表。
public ArrayList(int initialCapacity)
//构造一个初始容量为10的空列表。
public ArrayList()
'' //构造一个包含指定 collection 的元素的列表
public ArrayList(Collection<?  extends E> c)
```

Vector有四个构造方法：

```
public Vector()//使用指定的初始容量和等于零的容量增量构造一个空向量。
public Vector(int initialCapacity)//构造一个空向量，使其内部数据数组的大小，其标准容量增量为零。
public Vector(Collection<? extends E> c)//构造一个包含指定 collection 中的元素的向量
```

```
public Vector(int initialCapacity,int
capacityIncrement)//使用指定的初始容量和容量增量构造一个空的向量
```

ArrayList和Vector都是用数组实现的，主要有这么三个区别：

- Vector是多线程安全的，线程安全就是说多线程访问同一代码，不会产生不确定的结果。而ArrayList不是，这个可以从源码中看出，Vector类中的方法很多有synchronized进行修饰，这样就导致了Vector在效率上无法与ArrayList相比；
- 两个都是采用的线性连续空间存储元素，但是当空间不足的时候，两个类的增加方式是不同。
- Vector可以设置增长因子，而ArrayList不可以。
- Vector是一种老的动态数组，是线程同步的，效率很低，一般不赞成使用。

适用场景分析：

- Vector是线程同步的，所以它也是线程安全的，而ArrayList是线程异步的，是不安全的。如果不考虑到线程的安全因素，一般用ArrayList效率比较高。
- 如果集合中的元素的数目大于目前集合数组的长度时，在集合中使用数据量比较大的数据，用Vector有一定的优势。

HashMap 和 Hashtable 的区别

- HashMap去掉了HashTable的contains方法，但是加上了containsValue（）和containsKey（）方法。
- HashTable同步的，而HashMap是非同步的，效率上比HashTable要高。
- **HashMap允许空键值**，而HashTable不允许。

注意：*TreeMap*：非线程安全基于红黑树实现。*TreeMap*没有调优选项，因为该树总处于平衡状态。*Treemap*：适用于按自然顺序或自定义顺序遍历键*(key)*。

HashSet 和 HashMap 区别

- set是线性结构，set中的值不能重复，hashset是set的hash实现，hashset中值不能重复是用hashmap的key来实现的。
- map是键值对映射，可以空键空值。HashMap是Map接口的hash实现，key的唯一性是通过key值hash值的唯一来确定，value值是则是链表结构。

他们的共同点都是*hash*算法实现的唯一性，他们都不能持有基本类型，只能持有对象

线程

创建线程的方式及实现

Java中创建线程主要有三种方式：

1. 继承Thread类创建线程类

- 定义Thread类的子类，并**重写该类的run方法**，该*run*方法的方法体就代表了线程要完成的任务。因此把*run()*方法称为执行体。
- 创建Thread子类的实例，即创建了线程对象。
- 调用线程对象的start()方法来启动该线程。

```
package com.thread;
public class RunnableThreadTest implements Runnable{
```

```
	private int i;
	public void run(){
		for(i = 0;i <100;i++){

System.out.println(Thread.currentThread().getName()+"
"+i);
		}
	}
	public static void main(String[] args){
		for(int i = 0;i < 100;i++){

System.out.println(Thread.currentThread().getName()+"
"+i);
				if(i==20){
					RunnableThreadTest rtt = new
RunnableThreadTest();
					new Thread(rtt,"新线程
1").start();
					new Thread(rtt,"新线程
2").start();
				}
		}
	}
}
```

1. 通过Runnable接口创建线程类

- 定义runnable接口的实现类，并重写该接口的run()方法，该run()方法的方法体同样是该线程的线程执行体。
- 创建 Runnable实现类的实例，并依此实例作为Thread的target来创建Thread对象，该Thread对象才是真正的线程对象。
- 调用线程对象的start()方法来启动该线程。

```
package com.thread;
```

```
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.FutureTask;

public class CallableThreadTest implements
Callable<Integer>{
   public static void main(String[] args){
       CallableThreadTest ctt = new
CallableThreadTest();
       FutureTask<Integer> ft = new FutureTask<>(ctt);
       for(int i = 0;i < 100;i++){

System.out.println(Thread.currentThread().getName()+" 的
循环变量i的值"+i);
           if(i==20){
               new Thread(ft,"有返回值的线程").start();
           }
       }
       try{
           System.out.println("子线程的返回值: "+ft.get());
       } catch (InterruptedException e){
           e.printStackTrace();
       } catch (ExecutionException e){
           e.printStackTrace();
       }
   }
   @Override
   public Integer call() throws Exception{
       int i = 0;
       for(;i<100;i++){

System.out.println(Thread.currentThread().getName()+"
"+i);
       }
       return i;
   }
```

```
}
```

1. 通过Callable和Future创建线程

- （1）创建Callable接口的实现类，并实现call()方法，该call()方法将作为线程执行体，并且有返回值。
- （2）创建Callable实现类的实例，使用FutureTask类来包装Callable对象，该FutureTask对象封装了该Callable对象的call()方法的返回值。
- （3）使用FutureTask对象作为Thread对象的target创建并启动新线程。
- （4）调用FutureTask对象的get()方法来获得子线程执行结束后的返回值

```
package com.thread;

import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.FutureTask;
public class CallableThreadTest implements
Callable<Integer>{
public static void main(String[] args){
CallableThreadTest ctt = new CallableThreadTest();
FutureTask<Integer> ft = new FutureTask<>(ctt);
for(int i = 0;i < 100;i++) {
System.out.println(Thread.currentThread().getName()+" 的
循环变量i的值"+i);
if(i==20){
new Thread(ft,"有返回值的线程").start();
}
}
try {
System.out.println("子线程的返回值：
"+ft.get());
} catch (InterruptedException e) {
e.printStackTrace();
} catch (ExecutionException e){
```

```
e.printStackTrace();
}
}
@Override
public Integer call() throws Exception{
int i = 0;
for(;i<100;i++){
System.out.println(Thread.currentThread().getName()+"
"+i);
}
return i;
}
}
```

线程安全问题

线程安全是指要控制多个线程对某个资源的有序访问或修改，而在这些线程之间没有产生冲突。

在Java里，线程安全一般体现在两个方面：

- 多个thread对同一个java实例的访问（read和modify）不会相互干扰，它主要体现在关键字synchronized。如ArrayList和Vector，HashMap和Hashtable（后者每个方法前都有synchronized关键字）。如果你在interator一个List对象时，其它线程remove一个element，问题就出现了。
- 每个线程都有自己的字段，而不会在多个线程之间共享。它主要体现在java.lang.ThreadLocal类，而没有Java关键字支持，如像static、transient那样。

线程的生命周期

新建(New)、就绪（Runnable）、运行（Running）、 阻塞(Blocked)和死亡(Dead)5种状态：

1. 生命周期的五种状态

- 新建（new Thread）
  当创建Thread类的一个实例（对象）时，此线程进入新建状态（未被启动）。
  例如：`Thread t1=new Thread();`
- 就绪（runnable）
  线程已经被启动，正在等待被分配给CPU时间片，也就是说此时线程正在就绪队列中排队等候得到CPU资源。例如：`t1.start();`
- 运行（running）
  线程获得CPU资源正在执行任务（run()方法），此时除非此线程自动放弃CPU资源或者有优先级更高的线程进入，线程将一直运行到结束。
- 死亡（dead）
  当线程执行完毕或被其它线程杀死，线程就进入死亡状态，这时线程不可能再进入就绪状态等待执行。
- 自然终止：正常运行run()方法后终止。
- 异常终止：调用stop()方法让一个线程终止运行。
- 堵塞（blocked）
  由于某种原因导致正在运行的线程让出CPU并暂停自己的执行，即进入堵塞状态。
- 正在睡眠：用sleep(long t) 方法可使线程进入睡眠方式。一个睡眠着的线程在指定的时间过去可进入就绪状态。
- 正在等待：调用wait()方法。（调用motify()方法回到就绪状态）

- 被另一个线程所阻塞：调用suspend()方法。（调用resume()方法恢复）

线程方法 sleep()、join（）、yield（）有什么区别?

1. sleep()方法

   在指定的毫秒数内让当前正在执行的线程休眠（暂停执行），此操作受到系统计时器和调度程序精度和准确性的影响。 让其他线程有机会继续执行，但它并不释放对象锁。也就是如果有Synchronized同步块，其他线程仍然不能访问共享数据。注意该方法要捕获异常比如有两个线程同时执行(没有Synchronized)，一个线程优先级为MAX*PRIORITY*，另一个为*MIN*PRIORITY，如果没有Sleep()方法，只有高优先级的线程执行完成后，低优先级的线程才能执行；但当高优先级的线程sleep(5000)后，低优先级就有机会执行了。总之，sleep()可以使低优先级的线程得到执行的机会，当然也可以让同优先级、高优先级的线程有执行的机会。

2. yield()方法

   yield()方法和sleep()方法类似，也不会释放“锁标志”，区别在于，它没有参数，即yield()方法只是使当前线程重新回到可执行状态，所以执行yield()的线程有可能在进入到可执行状态后马上又被执行，另外yield()方法只能使同优先级或者高优先级的线程得到执行机会，这也和sleep()方法不同。

3. join()方法

   Thread的非静态方法join()让一个线程B“加入”到另外一个线程A的尾部。在A执行完毕之前，B不能工作。

```
Thread t = new MyThread();
t.start();
```

```
t.join();
```

保证当前线程停止执行，直到该线程所加入的线程完成为止。然而，如果它加入的线程没有存活，则当前线程不需要停止。

线程池

线程池的几种方式：

- newFixedThreadPool(int nThreads)

  创建一个固定长度的线程池，每当提交一个任务就创建一个线程，直到达到线程池的最大数量，这时线程规模将不再变化，当线程发生未预期的错误而结束时，线程池会补充一个新的线程
- newCachedThreadPool()

  创建一个可缓存的线程池，如果线程池的规模超过了处理需求，将自动回收空闲线程，而当需求增加时，则可以自动添加新线程，线程池的规模不存在任何限制
- newSingleThreadExecutor()

  这是一个单线程的Executor，它创建单个工作线程来执行任务，如果这个线程异常结束，会创建一个新的来替代它；它的特点是能确保依照任务在队列中的顺序来串行执行
- newScheduledThreadPool(int corePoolSize)

  创建了一个固定长度的线程池，而且以延迟或定时的方式来执行任务，类似于Timer。

Redis 有哪些类型

Redis支持五种数据类型：string（字符串），hash（哈希），list（列表），set（集合）及zset(sorted set：有序集合)。

内存中的栈(stack)、堆(heap)和静态区(static area)的说明

通常我们定义一个基本数据类型的变量，一个对象的引用，还有就是函数调用的现场保存都使用内存中的**栈空间**；而通过new关键字和构造器创建的对象放在**堆空间**；程序中的字面量（literal）如直接书写的100、"hello"和常量都是放在**静态区**中。

栈空间操作起来最快但是栈很小，通常大量的对象都是放在堆空间，理论上整个内存没有被其他进程使用的空间甚至硬盘上的虚拟内存都可以被当成堆空间来使用。

```
String str = new String("hello");
```

上面的语句中变量str放在栈上，用new创建出来的字符串对象放在堆上,而“hello"这个字面量放在静态区。